The OBLAS

Objective basic linear algebra subprograms

(One call for all LAS)

Dr. Craig C. Douglas

IBM T. J. Watson Research Center (Math)
Yale University (Computer Science)
CERFACS (Parallel Algorithms Group)

Web pages (English, French, or German)

http://www.cs.yale.edu/users/douglas-craig/
http://www.cerfacs.fr/~douglas/

§1. Motivation

Consider the following Fortran–90 intrinsic function:

$$A \;=\; matmul(B, C)$$

In the xlf90 implementation, there are 116 possible combinations of data types to deal with. There is 1 template which is compiled in 116 different ways to implement matmul.

This is eay to maintain (no bug reports since it was released) and extremely convenient for Fortran–90 users.

This philosophy was used earlier in the code *gemmw* on netlib and the multigrid package *madpack5* on MGNet (casper.cs.yale.edu or ftp.cerfacs.fr).

I use the BLAS in applications, not just linear algebra libraries. These run on 1 or more processors (made by a variety of companies).

The BLAS promotes portability *only* between machines with the same word size conventions.

On a $\boxed{\begin{array}{c}\text{RS6000}\\ \text{Cray}\end{array}}$ I use $\boxed{\begin{array}{c}\textbf{D}\text{xxxxx}\\ \textbf{S}\text{xxxxx}\end{array}}$ routines.

Not all computers come with a complete set of BLAS, nor both single and double precision floating point hardware.

Worse, the arguments for similar functions in the PBLAS and sparse BLAS are quite different from the BLAS.

So, who are you going to call when you need a truly portable interface to the various BLAS?

§2. OBLAS Overview

The basic idea is to *register* objects in a very simple database which is globally available to the OBLAS. Accessing this must be quick and it should be small.

Then calls to LAS (B, P, sparse, or some future ones) just pass registration numbers (a natural number to be precise) and almost nothing else to *wrapper routines* which call the correct LAS (via jump tables) with all of the correct arguments.

This allows the LAS developers the opportunity to add whatever they like in the future. It decreases the amount of information that must be passed around globally in a user's program. It decreases the number of LAS names that the user must memorize and the list of arguments become more obvious.

What could be more simple and obvious?

§2.1 Registration

$$
\begin{array}{l}
integer\ rx, obregister, whatx, typex, sizex, \\
\qquad procs, descx(*), ix, jx, \\
\qquad ldx, rowsx, colsx, incx, subdiags, superdiags \\
character * (*)\ uplo, diag \\
declaration\ for\ x \\
\qquad \vdots \\
rx = obregister(whatx, x, typex, sizex, \\
\qquad incx, ldx, rowsx, colsx, \\
\qquad procs, descx, ix, jx, \\
\qquad uplo, diag, subdiags, superdiags)
\end{array}
$$

$whatx$	Datum is a
0	Vector
1	General matrix
2	Symmetric matrix
3	Hermitian matrix
4	Triangular matrix
...	Sparse matrix types and future additions
+100	Band matrix
+200	Packed matrix

I use either m4 or cpp to get around the next can of worms...

$typex$	Datum is
1	real
2	complex

$sizex$	Datum word length is
1	single precision
2	double precision
$-sizeof(float)$	single precision
$-sizeof(double)$	double precision

A routine $obsizeof$ returns the sizeof values that can be employed directly in the $obregister$ calls, e.g.,

$$
\begin{array}{l}
integer\ obsingle, obdouble \\
\cdots \\
call\ obsizeof(obsingle, obdouble)
\end{array}
$$

Some registration-like routines are

Fn.	Subprogram	Functionality
×	*obregister*	Register general object
×	*obregister*1	Register object (1 processor only)
×	*obregisterp*	Register object (p processors only)
×	*obregisters*	Register object (sparse BLAS only)
×	*obreplicate*	Register object similar to another
×	*obreregister*	Re-register general object
×	*obdelete*	Remove an object from registry
×	*obnewaddress*	Update object's address
×	*obremap*	Change parallel coordinates
×	*obnewincrement*	Change increment
	obsizeof	Get size of float and double

While most of the properties of an object remain fixed during a program, some change and there are always exceptions. Examples include the following:

- An object is on a stack and its address changes every time a routine is entered.

- The increment of an object changes.

- So many things changed about an object that it is worth redeclaring everything in 1 call to the OBLAS database manager instead of almost everything in several calls.

LAS	$Routine\ call$
$BLAS$	$caxpy(n, a, x, incx, y, incy)$ $daxpy(n, a, x, incx, y, incy)$ $saxpy(n, a, x, incx, y, incy)$ $zaxpy(n, a, x, incx, y, incy)$
$PBLAS$	$pcaxpy(n, a, x, ix, jx, descx, incx, y, iy, jy, descy, incy)$ $pdaxpy(n, a, x, ix, jx, descx, incx, y, iy, jy, descy, incy)$ $psaxpy(n, a, x, ix, jx, descx, incx, y, iy, jy, descy, incy)$ $pzaxpy(n, a, x, ix, jx, descx, incx, y, iy, jy, descy, incy)$

or

$$call\ obaxpy(n, a, rx, ry)$$

Note the normal BLAS arguments are only present in the BLAS.

The OBLAS allow a number of extensions to the LAS that would be painful or a nightmare using the normal naming scheme and subprogram design.

It would be trivial to support mixed data types as an extension to all of the LAS (e.g., rx represents a real, single precision object and ry represents a complex, double precision object).

§2.2 Level 1 OBLAS

Subprogram call	Functionality
$obswap(n, rx, ry)$	$x \leftrightarrow y$
$obscal(n, alpha, rx)$	$x \leftarrow \alpha x$
$obcopy(n, rx, ry)$	$y \leftarrow x$
$obaxpy(n, alpha, rx, ry)$	$y \leftarrow \alpha x + y$
$obdot(n, dot, rx, ry)$	$dot \leftarrow x^T y$
$obdotc(n, dotc, rx, ry)$	$dot \leftarrow x^H y$
$obnrm2(n, nrm2, rx)$	$nrm2 \leftarrow \|x\|_2$
$obasum(n, asum, rx)$	$asum \leftarrow \|re(x)\|_1 + \|im(x)\|_1$
$obamax(n, amax, indx, rx)$	$indx \leftarrow 1^{st}\ k \ni \|re(x_k)\| + \|im(x_k)\| =$
	$max(\|re(x_i)\| + \|im(x_i)\|) = amax,\ \forall i$
$obrot(n, rx, ry, c, s)$	Apply plane rotation
$obrotg(rx, ry, c, s)$	Generate plane rotation

This follows the Level 1 BLAS fairly closely.

It does eliminate the first letter(s) syndrome from applications, but that would be easy anyway.

Question: Should n be eliminated from these calls? (I think so.)

§2.3 Level 2 OBLAS

Define

$$\begin{array}{rcl} op(A) & = & A,\ A^T,\ or\ A^H \\ op^{-1}(A) & = & A^{-1},\ A^{-T},\ or\ A^{-H} \end{array}$$

Subprogram call	Functionality
$obgemv(trans, alpha, ra, rx, beta, ry)$	$y \leftarrow op(A)x + \beta y$
$obhemv(alpha, ra, rx, beta, ry)$	$y \leftarrow \alpha Ax + \beta y$
$obsymv(alpha, ra, rx, beta, ry)$	$y \leftarrow \alpha Ax + \beta y$
$obtrmv(trans, ra, rx)$	$y \leftarrow \alpha op(A)x$
$obtrsv(trans, ra, rx)$	$y \leftarrow \alpha op^{-1}(A)x$
$obger(alpha, rx, ry, ra)$	$A \leftarrow \alpha xy^T + A$
$obgeru(alpha, rx, ry, ra)$	$A \leftarrow \alpha xy^T + A$
$obgerc(alpha, rx, ry, ra)$	$A \leftarrow \alpha xy^H + A$
$obher(alpha, rx, ra)$	$A \leftarrow \alpha xx^H + A$
$obher2(alpha, rx, ry, ra)$	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$
$obsyr(alpha, rx, ry, ra)$	$A \leftarrow \alpha xx^T + A$
$obsyr2(alpha, rx, ry, ra)$	$A \leftarrow \alpha xy^T + \alpha yx^T + A$
or	
$obmv(trans, alpha, ra, rx, beta, ry)$	$y \leftarrow op(A)x + \beta y$
$obtmv(trans, ra, rx)$	$y \leftarrow \alpha op(A)x$
$obtsv(trans, ra, rx)$	$y \leftarrow \alpha op^{-1}(A)x$
$obger(alpha, rx, ry, ra)$	$A \leftarrow \alpha xy^T + A$
$obgerc(alpha, rx, ry, ra)$	$A \leftarrow \alpha xy^H + A$
$obshr(alpha, rx, ra)$	$A \leftarrow \alpha xx^H + A$
$obshr2(alpha, rx, ry, ra)$	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$

Matrices come in many formats (dense, band, packed) for general, symmetric, Hermitian, and triangular. $obmv$ can call 20 BLAS and 8 PBLAS (but only one per call to $obmv$).

§2.4 Level 3 OBLAS

Define

$$side(A, B) \;=\; AB \; or \; BA.$$

Subprogram call	Functionality
$obgemm(transa, transb, alpha, ra, rb, beta, rc)$	$C \leftarrow \alpha op(A)op(B) + \beta C$
$obhemm(side, alpha, ra, rb, beta, rc)$	$C \leftarrow \alpha side(A, B) + \beta C$
$obsymm(side, alpha, ra, rb, beta, rc)$	$C \leftarrow \alpha side(A, B) + \beta C$
$obherk(trans, alpha, ra, beta, rc)$	$C \leftarrow \alpha side(A, A^H) + \beta C$
$obsyrk(trans, alpha, ra, beta, rc)$	$C \leftarrow \alpha side(A, A^T) + \beta C$
$obherk2(trans, alpha, ra, rb, beta, rc)$	$C \leftarrow \alpha AB^H + \alpha BA^H + \beta C$ or $C \leftarrow \alpha A^H B + \alpha B^H A + \beta C$
$obserk2(trans, alpha, ra, rb, beta, rc)$	$C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$ or $C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$
$obtran(alpha, ra, beta, rc)$	$C \leftarrow \alpha A^T + \beta C$
$obtranc(alpha, ra, beta, rc)$	$C \leftarrow \alpha A^H + \beta C$
$obtrmm(side, transa, alpha, ra, rc)$	$C \leftarrow \alpha side(op(A), C)$
$obtrsm(side, transa, alpha, ra, rc)$	$C \leftarrow \alpha side(op^{-1}(A), C)$
or	
$obmm(transa, transb, alpha, ra, rb, beta, rc)$	$C \leftarrow \alpha op(A)op(B) + \beta C$
$obtmm(side, transa, alpha, ra, rc)$	$C \leftarrow \alpha side(op(A), C)$
$obtsm(side, transa, alpha, ra, rc)$	$C \leftarrow \alpha side(op^{-1}(A), C)$
$obrk(trans, alpha, ra, beta, rc)$	$C \leftarrow \alpha side(A, A^H) + \beta C$
$obrk2(trans, alpha, ra, rb, beta, rc)$	$C \leftarrow \alpha AB^H + \alpha BA^H + \beta C$ or $C \leftarrow \alpha A^H B + \alpha B^H A + \beta C$
$obtran(alpha, ra, beta, rc)$	$C \leftarrow \alpha A^T + \beta C$
$obtranc(alpha, ra, beta, rc)$	$C \leftarrow \alpha A^H + \beta C$

The sparse toolkit of Heroux et al has different calling sequences for the same functionality based on the sparse matrix format. No offense meant, but **yuck**. The OBLAS fixes this.

3. Model implementations!

1. One in rather portable C. This supports programs calling it from C, C++, Fortran (77 and 90), and Pascal.

2. One in Fortran-90. This supports Fortran-90 and is making no effort to support callers from other languages. It uses function overloading, modules, and other *write only* coding styles adapted from C++.

3. One in C++ probable.

We expect to release each shortly after this workshop, hopefully having been improved by comments from the attendees.

Currently, the alternate Level 2 and 3 calling sequences are implemented.

The libraries are designed to be assembled in multiple ways:

1. BLAS only.

2. BLAS and PBLAS only.

3. BLAS, PBLAS, and a selection (or all) of the sparse BLAS.

This is to save space since a user does not necessarily want the entire BLACS plus either MPI or PVM added to their application on a single processor.

Restricting the number of data types automatically should be an option.

Who should use the OBLAS

There are 3 primary reasons why the OBLAS should be considered.

1. *Portability*. Codes conveniently run on machines with different word sizes and numbers of processors.

2. *Fewer routine names to remember*. There are too many LAS names already.

3. *Simplification*. The calls have fewer arguments which are consistent, which means that less information must be globally available.

The latter is particularly valuable for the sparse BLAS which has a different set of arguments for subprograms of the same functionality but different sparse matrix storage formats.

Who should avoid the OBLAS

1. *Benchmarkers*. Using the OBLAS will add an extra routine call per LAS call. It will also cause an extra library to be loaded, possible dynamically at run time the first time one of these routines is called.

2. *Librarians*. Extreme care must be exercised in order to avoid having potentially huge libraries (e.g., all of ScaLapack) loaded when only one data type is used.

3. *Limited memory users*. People using machines with virtually no memory per node may run out of memory unexpectedly with these routines.

Just as there are still a few justifiable instances of programming a computer in assembly language, there are still real situations where programming the BLAS, PBLAS, or sparse BLAS directly are justifiable.

Comments needed

1. Should the OBLAS be quite compact or follow the BLAS directly? How much should the LAS be shrunk?

2. Which sparse BLAS or toolkit(s) should be used?

3. Do you want an error code returned instead of stopping the entire code. Do you want the error handling to be an option (stop or return a code)?

4. Is there a better registration methodology?

5. As the world turns (sic.) to Windows, does a mixed language implementation make sense?

6. Anything else...